Pergamon

Computers Math. Applic. Vol. 30, No. 8, pp. 33–39, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00135-2

A Cardinalised Binary Representation for Exponentiation

K.-Y. LAM, K. SUNG AND L. C.-K. HUI
Department of Information System & Computer Science
National University of Singapore, Republic of Singapore
lamky@iscs.nus.sg

(Received January 1995; accepted March 1995)

Abstract—This paper introduces the Cardinalised Binary Representation (CBR) of integers. A family of algorithms for converting binary integers to their CBR equivalents is described. Characteristics of the resulting representation is analyzed and exploited to achieve performance improvement for large integer exponentiation operations. This paper demonstrates that the CBR is a more general scheme than the well known Binary Redundant Representation (BRR) [1], and shows that both software and hardware CBR exponentiation algorithms operate more efficiently than that of BRR.

Keywords—Public key cryptography, Systolic arrays.

1. INTRODUCTION

The purposes of this paper are to introduce the Cardinalised Binary Representation (CBR) as a general representation for binary integers, and to demonstrate that the CBR may be exploited to improve the performance of square-and-multiply exponentiation operations in both software and hardware implementations. Exponentiation of large integers is the basis of several well-known cryptographic algorithms such as RSA [2] and El Gamal [3]. The calculations involved are complex, and can be time-consuming, especially when performed in software. As a result, algorithms which speed up implementations of modular exponentiation are of considerable practical significance; see, for example, [1,4,5].

The exponentiation problem is to compute $m^e \pmod N$, where m and N are huge integers, and e is an integer exponent. Performance of the square-and-multiply algorithm depends on the number of self-squaring and modular multiplication to be computed [6]. For an n-bit exponent e, n self-squaring is always needed, while the number of modular multiplications involved is determined by the number of nonzero digits (weight) in the representation of e. Note that throughout this paper, for simplicity, $m^j \pmod N$ is always written as m^j for any integer j.

Various representations of e have been suggested with the same goal of reducing the number of multiplications involved. An approach based on the Binary Redundant Representation (BRR) was proposed by Zhang (also independently discovered by Mitchell [7]), which requires an average of $n/3$ multiplications at the cost of precomputing m^{-1} ([1, p. 16]).

This paper describes the use of CBR to represent each integer exponent with fewer nonzero digits. It is a more flexible scheme than the BRR because, instead of limiting the recoded digits to $\{0, 1, \overline{1}\}$, CBR allows the recoded string to contain any cardinal. The general exponentiation algorithm for CBR-recoded exponents is also described. Due to practical considerations, the l-bounded CBR is defined to restrict the set of cardinals allowed in the recoded CBR string. The

We would like to thank the anonymous referees for their helpful comments.

nonzero digits on a l-bounded CBR must be an odd cardinal between 0 and $2^l - 1$. The $SS(l)$ algorithm converts a binary number into its l-bounded CBR equivalent [5]. It has been proven that $SS(l)$ is a minimum weight recoding algorithm, and that the expected number of nonzero digits in the resulting representation is $n/(l+1)$ [8]. In this way, with general l-bounded CBR, it is possible to implement an exponentiation operation that performs better than those reported in [1,7]. For example, from the expected weight of the recoded exponent it is observed that the slowest version of $SS(l)$ algorithms, i.e., $SS(2)$, is comparable to that of Zhang's BRR [1]. Further, as to be discussed, a linear systolic array system can be designed to implement the 2-bounded CBR exponentiation algorithm. This is achieved by analyzing and exploiting the features of 2-bounded CBR exponents. It is demonstrated that the systolic array implementation of 2-bounded CBR exponentiation is simpler and more efficient than that of BRR exponentiation.

2. THE CARDINALISED BINARY REPRESENTATION

DEFINITION 2.1. *Let $\mathcal{N}_0$ be the set of cardinals. A Cardinalised Binary Representation (CBR) of an integer e, $e \geq 0$, is a binary radix polynomial*

$$\sum_{i=0}^{n-1} e_i * 2^i, \qquad \text{where } \forall i \in [0..(n-1)], \quad e_i \in \mathcal{N}_0$$

such that $\sum_{i=0}^{n-1} e_i * 2^i$ is equal to e. In this paper, e is written as a digit string "$e_{n-1}e_{n-2}\ldots e_1e_0$" where e_{n-1} is the most significant digit. ∎

Note that, unlike the binary representation, the CBR is not unique. For example, $e = 11011 = 03003 = 10051$. The exponentiation algorithm to compute m^e, with e being in CBR with digit string "$e_{n-1}e_{n-2}\ldots e_1e_0$," is given in Figure 1.

```
                x ← 1;
                For i ← n − 1 Downto 0 Do
(Step A:)           x ← x^2;
(Step B:)           x ← x · m^{e_i};
                EndDo
```

Figure 1. The CBR exponentiation algorithm.

```
i ← 0;
While i ≤ n − 1 Do
    If (e_i == 0) Then
      i ←i + 1;
    Else /* (e_i == 1) */
      find the largest j ∈ [0..n − 1] s.t. (i + l > j ≥ i) And (e_j == 1);
      let e_j e_{j−1} ... e_i be the binary representation of the odd integer q;
      change e_i to the cardinal q;
      For j' ←i + 1 To j Do
          change e_{j'} to 0;
      EndDo
      i ←j + 1;
    EndIf
EndDo
/* The final e_{n−1} ... e_0 is the l-bounded CBR of e */
```

Figure 2. The l-bounded CBR recoding algorithm.

Observing *Step B* of Figure 1, for an efficient implementation, it is important to restrict the set of valid cardinals for e_i, so that precomputation of the associated m^{e_i} is possible.

DEFINITION 2.2. *An l-bounded CBR is a CBR such that if $e_{n-1}e_{n-2}\ldots e_1e_0$ is the CBR of e, then*

$$\forall i \in [0..(n-1)], \quad e_i = 0, \quad e_i = 1, \qquad \text{or } e_i = 2j+1 \qquad \text{for some } j \in \left[1..\left(2^{l-1}-1\right)\right].$$

That is, if e_i is nonzero, e_i is an odd cardinal between 1 and $2^l - 1$. ∎

In this respect, $SS(l)$ has been shown to be an efficient method to convert a binary representation to an l-bounded CBR [5]. It has been proven that, for a fixed l, if the $SS(l)$ conversion is by means of a left-to-right scan (which is more convenient for software implementation, refer to Figure 3) or a right-to-left scan (which is required by the hardware implementation, see Lemma 3.1), the number of nonzero digits in the resulting l-bounded CBR is minimal and is close to $n/(l+1)$ for large n [8].

The $SS(l)$ algorithm can be used to compute the exponentiation operation m^e by first translating the n-bit binary integer e into its CBR equivalent and then performing the square-and-multiply operations. A right-to-left version of the $SS(l)$ recoding algorithm is given in Figure 2.

```
x ← 1;   i ← n;
Loop{
    While (i > 0) And (e_{i-1} = '0')
        { i ← i − 1;   x ← x^2; }
    If (i ≤ 0) Then done and return x as answer;
    j ← i − l;
    While (j < 0) Or (e_j = '0')   { j ← j + 1; }
    a ← the value e_{i-1} ... e_{j+1};
    While (i > j)   { i ← i − 1;   x ← x^2; }
    x ← x · H[a];
} /* End Loop */
```

Figure 3. A $SS(l)$ exponentiation algorithm.

It is important to note that the recoding process and the square-and-multiply computation may be combined together when implemented in software. The $SS(l)$ algorithm, though very useful, is very simple. As an illustration, an exponentiation algorithm making use of the left-to-right version of it, is outlined in Figure 3. The algorithm assumes that the precomputed values are stored in a table $H[1..(2^{l-1}-1)]$ where $H[j]$ stores the value m^{2j+1}. It is obvious that, due to the number of self-squaring operations involved, the exponentiation algorithm runs in $O(n)$ time.

Zhang's BRR exponentiation algorithm [1] is comparable to that of the simplest version of $SS(l)$, $SS(2)$, because:

1. The expected weight in both recoded strings is $n/3$, where n is the number of bits in the original binary number.
2. The running time of both exponentiation algorithms is $O(n)$.
3. The recoded strings contain only three different digits ($\{0, 1, \bar{1}\}$ for BRR, and $\{0, 1, 3\}$ for CBR).
4. The recorded strings are guaranteed to not contain two consecutive nonzero digits for a right-to-left recoding (refer to Lemma 3.1).

For the above reasons, when implemented in software, $SS(2)$ has a similar performance as the algorithm proposed by Zhang [1]. However, the $SS(2)$ is more efficient because it does not require

the computation of m^{-1}. In addition, $SS(l)$ provides more flexibility since it allows implementors to further speed up the square-and-multiply computation at the cost of more precomputation steps, i.e., with larger l. This flexibility is of great importance if the exponentiation algorithm is to be used by public key systems that compute different exponents of a fixed base such as the El Gamal scheme [3].

3. SYSTOLIC ARRAY IMPLEMENTATION

In this section, it is demonstrated that a right-to-left recoded 2-bounded CBR integer carries the same characteristic as a BRR in that both representations prevent the occurrence of two consecutive nonzero digits. Based on this characteristic, an efficient linear systolic array is designed to implement 2-bounded CBR exponentiation algorithm.

LEMMA 3.1. *Given an n-bit exponent e, any two consecutive digits of the right-to-left 2-bounded CBR of e must be in the set $\{00, 01, 03, 10, 30\}$.*

PROOF. We prove by induction on n, the length of e, for $n \geq 2$.

Basis case: When $n = 2$, the possible values of e in 2-bounded CBR are: $\{00, 01, 10, 03\}$. Hence, the lemma is true for $n = 2$.

Induction case: Assume that the lemma is true for $n = k$. Consider an exponent e' of length $k+1$ (denoted $e'_k \ldots e'_0$), since CBR is a right-to-left recoding scheme, if the recoding process is applied to the rightmost k bits of e', i.e., $e'_{k-1} \ldots e'_0$, the resulting string will not have two consecutive nonzero digits by assumption. Now we have two cases:

(i) $e'_k = 0$, and

(ii) $e'_k = 1$.

In case (i), the lemma is trivially true. We therefore consider case (ii). By assumption, the two leftmost digits of the 2-bounded CBR $e'_{k-1} \ldots e'_0$ must be in the set $\{00, 01, 10, 03, 30\}$. In fact, $e'_{k-1}e'_{k-2}$ must be in $\{00, 01, 10, 03\}$ since CBR is a right-to-left recoding scheme; hence, digit 3 must be preceded by a 0. Therefore, if e'_k is included, the last three digits are in $\{100, 101, 110, 103\}$ and the set of possible 2-bounded CBR is $\{100, 101, 030, 103\}$. So the lemma is true for $n = k + 1$, and the proof is complete. ∎

From Lemma 3.1, it is guaranteed that if two 2-bounded CBR digits are processed simultaneously, the only possible string combinations are: $\{00, 01, 03, 10, 30\}$. Since there can only be one nonzero digit, it follows that *Step A* of Figure 1 will be executed twice and *Step B* of the same figure will be executed at most once. Table 1 summarizes the relationship of the digits of 2-bounded CBR and their corresponding exponentiation operations. This observation is essential for the efficient hardware implementation of the algorithm.

Table 1. Valid 2-bounded CBR digits and the exponentiation operations.

Valid $SS(2)$ Digits	Required Operations	
0	$x := x \cdot x$	*(Step A)*
1	$x := x \cdot x$	*(Step A)*
	$x := x \cdot m$	*(Step B)*
3	$x := x \cdot x$	*(Step A)*
	$x := x \cdot m^3$	*(Step B)*

Assuming n is even, a linear systolic array of $n/2$ identical processing stages can be implemented to compute the 2-bounded *CBR Exponentiation Algorithm*. Referring to Figure 4, each processing stage consists of three identical Processing Elements (PEs), each of which is controlled by two input signals. The control signals α_j^k and β_j^k of Figure 4 are defined in Table 2, where k is an integer in [1..3], while $j \in [0..(n-2)]$ is an even integer such that α_j and β_j are used at stage $(n-$

$j)/2$. Note that α_j and β_j are generated from the digits $e_j e_{j+1}$. Also, as illustrated in Figure 5, each PE is basically a simple integer multiplication unit (all operations are modular arithmetics) with an internal multiplexor to route one of the input channels to the actual multiplication unit. Three PEs are sufficient for each stage because from Lemma 3.1, it is guaranteed that a maximum of three operations are required to process two digits of a 2-bounded CBR-recoded exponent. Since each processing stage of the systolic array consumes two digits and there are n digits in the 2-bounded CBR-recoded exponent, only $n/2$ stages are required. There are three PEs in each stage, so the startup time of the pipeline is $3n/2$. The product of time and area is $O(9n^2/4)$.

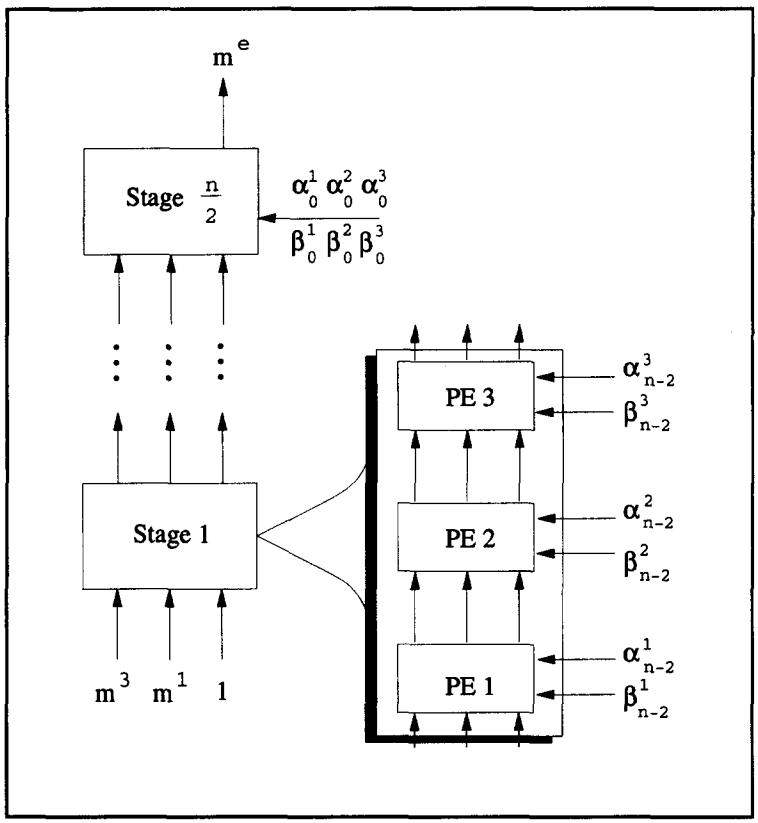

Figure 4. A linear systolic array for 2-bounded CBR exponentiation.

Table 2. Control signals for 2-bounded CBR exponentiation.

$e_j e_{j+1}$	$\alpha_j^1 \beta_j^1$ *(PE1)*	$\alpha_j^2 \beta_j^2$ *(PE2)*	$\alpha_j^3 \beta_j^3$ *(PE3)*
0 0	1 1 *(PE1: No-Op)*	0 0 *(PE2: Step A)*	0 0 *(PE3: Step A)*
0 1	0 0 *(PE1: Step A)*	0 0 *(PE2: Step A)*	0 1 *(PE3: Step B, m^1)*
0 3	0 0 *(PE1: Step A)*	0 0 *(PE2: Step A)*	1 0 *(PE3: Step B, m^3)*
1 0	0 0 *(PE1: Step A)*	0 1 *(PE2: Step B, m^1)*	0 0 *(PE3: Step A)*
3 0	0 0 *(PE1: Step A)*	1 0 *(PE2: Step B, m^3)*	0 0 *(PE3: Step A)*

The design of Figure 4 is similar to Zhang's linear systolic array for the Binary Redundant Representation *(BRR)* exponentiation algorithm [1]. However, with the BRR recoding scheme, Zhang's system must include an additional stage to cater for the extra digit which may be generated by the BRR recoding scheme. More importantly, the 2-bounded CBR algorithm has replaced m^{-1} by m^3. As depicted in Figure 6, m^3 can be computed with two PEs. This

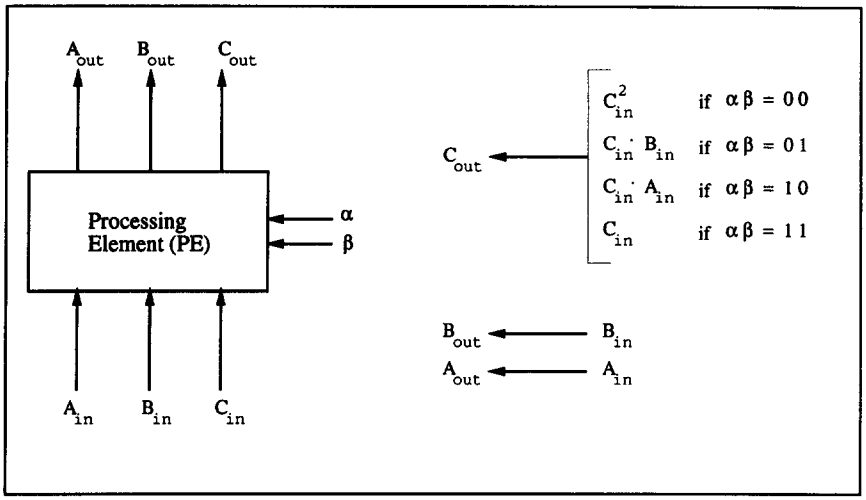

Figure 5. The functional diagram of a PE.

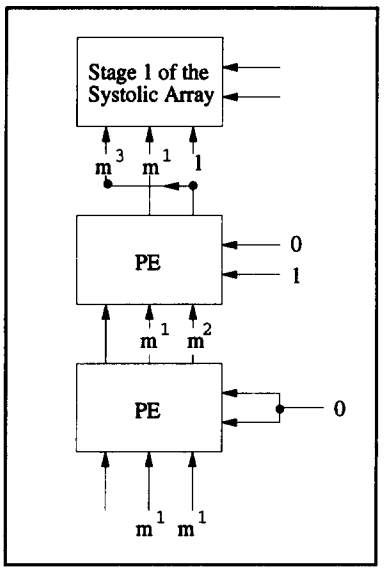

Figure 6. Initialization stage for 2-bounded CBR exponentiation.

significantly simplifies the initialization time and resources in that no specially designed hardware is required by the initialization stage.

4. CONCLUSION

We have defined the Cardinalised Binary Representation (CBR) as a more general scheme for encoding binary integers. The major strength of the CBR approach is in its ability to encode up to l bits of information into a single cardinal digit. We have demonstrated that the efficiency of a 2-bounded CBR exponentiation is comparable to that of the BRR exponentiation. The BRR approach requires the computation of m^{-1}, while the 2-bounded CBR requires m^3. For most computer systems, it is typically more efficient to compute m^3 than m^{-1}. As explained, a hardware system can generate the required m^3 with two basic processing elements. Although it is possible to implement fast hardware inversion [9], it would mean extra specialized hardware. Consequently, the initialization stage of the 2-bounded CBR systolic array implementation is much more efficient in terms of both processing time and computing resources. Also, in general, the BRR exponentiation algorithm needs to cater for the extra digit resulting from the recoding process. The CBR approach does not suffer from this problem.

The most important advantage of the CBR approach is that it is possible to trade complexity for computation time. For example, it is possible to design a linear systolic array to compute exponentiation with 3-bounded CBR-recoded exponents. In this case, there would be more inputs into each PE (increase in circuitry complexity) but with only $O(4n/3)$ stages (decrease in computation time). We are currently investigating the issues involved in hardware implementation of l-bounded CBR exponentiation algorithms for $l \geq 3$.

REFERENCES

1. C.N. Zhang, An improved binary algorithm for RSA, *Computers Math. Applic.* **25** (6), 15–24 (1993).
2. R.L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21**, 120–126 (1978).
3. T. El Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. on Information Theory* **31**, 469–472 (1985).
4. A. Selby and C.J. Mitchell, Algorithms for software implementations of RSA, In *Proceedings of the IEE*, Part E, pp. 166–170, (1989).
5. L.C.K. Hui and K.Y. Lam, Fast square-and-multiply exponentiation for RSA, *Electronics Letters* **30** (17), 1396–1397 (1994).
6. D.E. Knuth, *The Arts of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd Edition, Addison-Wesley, Reading, MA, (1981).
7. J. Jedwab and C.J. Mitchell, Minimum weight modified signed-digit representations and fast exponentiation, *Electronics Letters* **25**, 1171–1172 (1989).
8. K.Y. Lam and L.C.K. Hui, On the efficiency of $SS(l)$ square-and-multiply exponentiation algorithm, *Electronics Letters* (to appear).
9. D.Y.Y. Yun and C.N. Zhang, A fast carry-gree algorithm and hardware design for extended integer GCD computation, Presented at the *International ACM Conference on Symbolic and Algebra Computation*, Waterloo, Canada, July 1986, pp. 82–84.